VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace adaptivních algoritmů

Implementation of Adaptive Algorithms

2016

Matěj Kahánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Matěj Kahánek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace adaptivních algoritmů
Implementation of Adaptive Algorithms

Jazyk vypracování: čeština

Zásady pro vypracování:

Prostudujte a popište problematiku adaptivní filtrace včetně vybraných typů adaptivních algoritmů. Prozkoumejte možnosti návrhu a implementace adaptivních filtrů na bázi virtuální instrumentace v programovacím jazyce G pomocí NI LabVIEW Adaptive Filter Toolkit. Vybrané adaptivní algoritmy implementujte v některém textově orientovaném programovacím jazyce např.: C, C++, C#, apod., ve formě dynamicky linkované knihovny (DLL). V prostředí LabVIEW vytvořte toolkit vlastních subVI, které budou používat pro adaptivní filtraci metody implementované v této DLL knihovně. Vlastní implementace algoritmů srovnejte s algoritmy dostupnými v Adaptive Filter Toolkit. Charakterizujte aplikace, které lze pomocí tohoto toolkitu řešit. Pro zvolené aplikace navrhněte a realizuje virtuální přístroje, které budou využitelné pro výukové účely.

Úkoly:
1. Teoretický rozbor základních poznatků z oblasti adaptivní filtrace včetně detailního popisu vybraných adaptivních algoritmů (např. LMS, NLMS, FBLMS, RLS, QR-RLS, apod.).
2. Podrobný popis NI LabVIEW Adaptive Filter Toolkit včetně charakteristiky funkcí z paletové nabídky Adaptive Filters. Analýza reálných aplikací, které je možno řešit s využitím NI LabVIEW Adaptive Filter Toolkit.
3. Implementace dynamicky linkované knihovny (knihoven) pro zvolené adaptivní algoritmy v některém z textově orientovaných programovacích jazyků např.: C, C++, C#, apod.
4. Vytvoření toolkitu pro prostředí LabVIEW využívajícího vlastní knihovnu (knihovny) pro adaptivní filtraci.
5. Návrh a realizace virtuálních přístrojů pro vybrané aplikace adaptivní filtrace (např. adaptivní potlačování šumu, adaptivní ekvalizace přenosového kanálu, apod.).
6. Tvorba podrobné dokumentace k vytvořeným aplikacím, tak aby je bylo možné využít pro výukové účely.

Seznam doporučené odborné literatury:

[1] JAN, Jiří. Číslicová filtrace, analýza a rekonstrukce signálů. 2. vyd. Brno: VUTIUM, 2002, 428 s. ISBN 80-214-1558-4.
[2] HAYKIN, Simon. Adaptive Filter Theory. 4. vyd. Prentice Hall, 2001, 936 s. ISBN 978-0130901262.
[3] Manuals - LabVIEW 2013 Adaptive Filter Toolkit Help.
[4] SMÉKAL, Zdeněk. Číslicové zpracování signálů. Brno: VUT Ústav telekomunikací, 2004, 152 s.
[5] HAVLÍČEK, J., VLACH, J. a kol. - Začínáme s LabVIEW. BEN - technická literatura, Praha, 2008. ISBN 978-80-7300-245-9.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radek Martinek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 29. dubna 2016 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Rád bych na tomto místě poděkoval doktoru Radku Martinkovi, vedoucímu této práce, který mi s prací pomohl a poskytl mi potřebnou literaturu a podklady.

Abstrakt

Tato bakalářská práce se zabývá rozborem teorie adaptivních filtrů. Práce se dále zaměřuje na možnosti návrhu a implementace adaptivních filtrů na bázi virtuální instrumentace ve vývojovém prostředí LabVIEW od americké společnosti National Instruments. Praktická část je věnována vybraným adaptivním algoritmům, které jsou implementovány jak v LabVIEW, tak pomocí programovacího jazyka C# ve formě dynamicky linkovaných DLL knihoven. Na základě poznatků z implementace je provedeno srovnání obou možností. Pomocí softwarového nástroje National Instruments LabVIEW je v praktické části vytvořena řada virtuálních přístrojů, demonstrujících adaptivní filtraci.

Klíčová slova: Adaptivní filtry, adaptivní algoritmy, virtuální instrumentace, LabVIEW

Abstract

This bachelor's thesis deals with the analysis theory of adaptive filters. The study also deals with possibilitis of today's design and implementation of adaptive filters based on virtual instrumentation with National Instruments LabVIEW. The practical part includes the implementation of selected adaptive algorithms using LabVIEW and the programming language C# as a dynamically linked libraries DLLs. On the basis of the implementation is the comparison of both options. Using the software tool, National Instruments LabVIEW is created several virtual machines, demonstrating adaptive filtering.

Keywords: Adaptive filters, adaptive algorithms, virtual instrumentation, LabVIEW

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek a symbolů

- C# - Programovací jazyk
- DLL - Dynamic-Link Library - Dynamicky linkovaná knihovna
- EKG - Elektrokardiograf
- FBLMS - Fast Block LMS - Rychlý blokový LMS algoritmus
- FFT - Fast Fourier Transform - Rychlá Fourierova transformace
- FIR - Finite Impulse Response - Konečná impulsní charakteristika
- IFFT - Inverse Fast Fourier Transform - Inverzní rychlá Fourierova Transformace
- IIR - Infinite Impulse Response - Nekonečná impulsní charakteristika
- LabVIEW - Laboratory Virtual Instrument Engineering Workbench - Název programu společnosti National Instrumets
- LMS - Least Mean Square - Algoritmus nejmenší odchylky čtverců
- LPC - Linear predicted coding - Lineární predikční kódování
- NLMS - Normalized LMS - Normalizovaný LMS algoritmus
- QR dekompozice - Typ rozkladu matice
- QR-RLS - RLS s QR dekompozicí
- RLS - Recursive Least Square - Rekurzivní algoritmus nejmenších čtverců
- SNR - Signal to Noise Ratio - Odstup signálu od šumu
- VI - Virtual Instrument - Virtuální přístroj

1 Úvod

Filtr představuje systém popisující relaci mezi vstupním a výstupním signálem. Mnohdy je třeba odstranit nebo potlačit nežádoucí vlivy vnějšího prostředí, které narušují potřebnou kvalitu přenášeného signálu. Základní vlastností filtrů je potlačit nebo naopak zvýraznit určité kmitočtové složky vstupního signálů. Jedná se např. o odstranění střídavé či stejnosměrné složky signálu, omezení určitého frekvenčního pásma nebo snížení šumu resp. brumu. Filtrace se dnes využívá v nespočetném množství oborů jako je např. audiotechnika, radiotechnika, telekomunikační technika, elektronika nebo biomedicína. V praktické části práce jsou realizovány tyto reálné aplikace:

- Adaptivní potlačení echa.
- Adaptivní lineární predikce.
- Extrakce EKG signálu plodu.

Tato práce se konkrétně zabývá problematikou adaptivních filtrů, které mohou měnit své vlastnosti v průběhu měření a adaptovat se na změny vstupního signálu. Detailně popisuje základní zástupce adaptivních algoritmů a to LMS, NLMS, RLS a QR-RLS. Práce má za úkol seznámit s celkovou problematikou a poté aplikovat zjištěné poznatky v grafickém programovacím nástroji LabVIEW od firmy National Instruments, který umožňuje vytvořit virtuální měřící, testovací nebo řídící aplikace. Tento nástroj se dnes využívá při výuce také na Vysoké škole Báňské.

Pro adaptivní filtraci zde existuje rozšíření s názvem Adaptive Filter Toolkit, kde lze demonstrovat jednotlivé algoritmy. Cílem je toto rozšíření popsat a vytvořit několik cvičných úloh pro studenty. Všechny tyto algoritmy budou poté také implementovány pomocí programovacího jazyka C# a importovány v DLL knihovnách do programu LabVIEW.

2 Současný stav problematiky a cíle bakalářské práce

Adaptivní filtrace má dnes velké možnosti využití, vzhledem k roustoucí hardwarové výkonnosti počítačů, signálových zařízení a mobilních zařízení. Lze tedy využívat i komplikovanější adaptivní algoritmy v rozsáhlejších úlohách. V současné době se pro návrh adaptivních algortimů využívá vývojového prostředí Matlab [18], jazyk C [16] a LabVIEW [3], [4], [11], [5]. Vývojové prostředí LabVIEW je jedním z prostředí pro testování, měření a řízení aplikací. Největší předností LabVIEW je grafická podoba kódu, která výrazně usnadňuje implementaci aplikace. Pro tvorbu není třeba žádných speciálních znalostí prostředí ani programovacích jazyků.

LabVIEW nabízí také možnost připojení extérních přístrojů a hardwarových nástrojů a zakomponovat je do virtuálního přístroje. Další nespornou výhodou je množství rozšiřujících balíků, které dale rozšiřují oblast využití LabVIEW. Toto návrhové prostředí umožňuje simulovat a otestovat navržený přístroj před samotnou výrobou a minimalizova tak skryté chyby. Jednou z takových je právě rozšíření Adaptive Filter Toolkit [5], které bylo vytvořeno pro jednoduchou simulaci adaptivních filtrů a algoritmů.

V současné době neexistují žádné materiály a podklady k LabVIEW Adaptive Filter Toolkit psané v češtině. Prvním cílem této bakalářské práce je tedy seznámit čtenáře s vývojovým prostředím, jeho možnostmi a seznámit jej také se zmíněným rozšířením pro adaptivní filtry. Dalším cílem je vytvořit výukový materiál a sadu vzorových přkladů implementace, což může být využito k výukovým účelům a lepšímu pochopení problematiky adaptivních filtrů.

3 Adaptivní filtrace

Nastane-li situace, kdy je potřeba vytvořit filtr pracující v neznámém prostředí nebo prostředí měnící se v čase, nelze předem identifikovat zkreslení a tudíž ani navrhnout optimální řešení pro filtraci. Pro tyto situace je třeba vytvořit filtr adaptivní.

Tento filtr je schopen v daném prostředí v průběhu své práce sám získávat potřebné informace a tím reagovat na změny vlastností signálu. Dokáže se tedy adaptovat a zpracovat tak i signály, generované nestacionárními procesy, aniž by mu byly známy časově proměnné parametry těchto procesů. Pro správný chod adaptivních filtrů je jim nutno průběžně poskytovat dodatečné informace v podobě tzv. referenčního signálu. To vyžaduje zavedení dalšího vstupu kromě samotného pozorovaného signálu. Referenční signál do jisté míry souvisí s předpokládaným výstupním signálem, který určitým způsobem aproximuje. Mnohdy se jedná přímo o zmíněný výstupní signál.

Může se zdát, že je zbytečné filtrovat signál, jehož požadovaný výstup známe s dostatečnou přesností. Je však neustále nutné periodicky upravovat referenční signál k naučení adaptivního filtru odstranit zkreslení přenosové cesty, typicky u telekomunikačních sítí.

Základní dělení struktury adaptivních filtrů je na lineární a nelineární. V drtivé většině případů se používá pouze lineární verze, která je jednodušší a klade menší početní nároky.

Detailněji se tedy zaměříme pouze na lineární zastoupení filtrů. Tyto filtry rozdělujeme podle délky impulsní charakteristiky na následující:

- FIR filtr - filtr s konečnou impulsní charakteristikou.
- IIR filtr - filtr s nekonečnou impulsní charakteristikou.

3.1 Filtr typu FIR

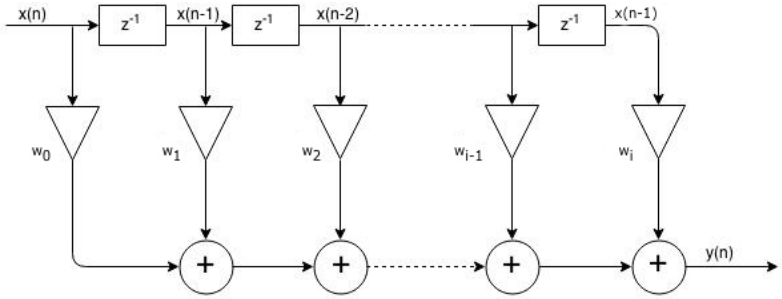
Jedná se o filtr s konečnou impulsní charakteristikou, někdy také nazýván jako nerekurzivní filtr bez zpětné vazby. Jeho výhodou je téměř naprostá stabilita a lineární fáze. Filtr typu FIR můžeme popsat pomocí Rov. 1, která představuje diferenční rovnici.

$$y(n) = \sum_{i=0}^{N-1} w_i x(n-i) = w(n) \cdot x(n), \tag{1}$$

kde $n = 1, 2, 3, ...$, $y(n)$ je výstupní signál, $x(n)$ je vstupní signál a w_i jsou koeficienty impulsní charakteristika filtru o délce N.

Z Rov. 1 můžeme snadno odvodit jeho základní schéma (Obr. 1).

Pro adaptivní systémy se koeficienty impulsní charakteristiky FIR filtru stanovují pomocí adaptivního algoritmu. Zde tedy nastává zásadní problém a to ten, že se z FIR filtru stává rekurzivní. Koeficienty adaptivního filtru jsou totiž přizpůsobovány podle výstupního signálu a tím ovlivňují celou filtraci.[1] Pokud nahradíme koeficienty impulsní charakteristiky FIR filtru koeficienty $\mathbf{w}$ adaptivního algoritmu, tedy tzv. vahami, upravíme Rov. 1 takto:



Obrázek 1: Blokové schéma FIR filtru

$$y(n) = \sum_{i=0}^{N-1} w_i x(n-i) = \mathbf{w} \cdot x(n). \tag{2}$$

Pro optimalizaci koeficientů adaptivního filtru můžeme stanovit vektor vah $\mathbf{w}$ a vektor vstupního signálu $\mathbf{x}()$, ze kterého je vybráno N vzorků, následnovně:

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_{N-1}], \tag{3}$$

$$\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-N+1)]^T, \tag{4}$$

kde N je délka frekvenční charakteristiky a zároveň délka vektorů. Vektor $\mathbf{x}()$ je transponovaný (sloupcový).

Nyní tedy můžeme Rov. 2 upravit podle Rov. 3 a 4 a použít v následujících rovnicích 5 a 6:

$$y(n) = \mathbf{w} \cdot \mathbf{x}^T(n), \tag{5}$$

$$e(n) = d(n) - \mathbf{w} \cdot \mathbf{x}^T(n), \tag{6}$$

kde $e(n)$ je chybový signál a $d(n)$ je referenční signál.

Tímto postupem jsme se vyhnuli použití konvoluce v původní Rov. 2 a nahradili ji součinem vektorů, přičemž vektor $\mathbf{w}$ je řádkový a vektor $\mathbf{x}^T(n)$ sloupcový. Takto získané vztahy jsou základem pro adaptivní filtraci. Z rovnic je také vidět důležitost správného stanovení délky impulsní charakteristiky filtru N. Tato délka ovlivňuje délku jednotilivých iterací a tím i časovou náročnost procesu filtrace.[1]

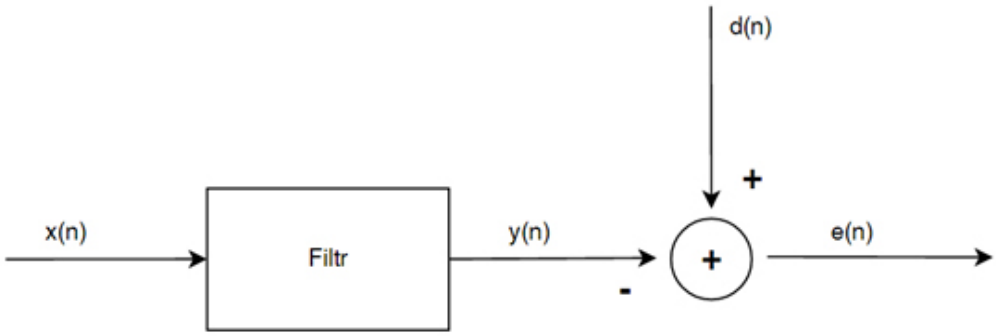
3.2 Filtr typu IIR

Filtry IIR mají nekonečnou impulsní charakteristiku, což je dáno zpětnou vazbou na výstupní signál. Oproti FIR filtrům mají menší výpočetní nároky a tím se zkracuje také čas výpočtů. Tato výhoda se však odráží v jeho stabilitě, která je výrazně menší než je tomu u

FIR fitru. Celkový návrh je složitější a obsahuje nelineární fázi. Ve zvolené problematice LabVIEW se však využívají pouze FIR filtry, proto IIR filtr nebude dále popisován.

3.3 Wienerův filtr

Adaptivní filtry jsou do jisté míry podobné k stacionárnímu Wienerovu filtru, avšak mohou měnit své koeficienty v čase tak, aby dokázaly efektivně potlačit nežádoucí změny na vstupním signálu.[2]



Obrázek 2: Blokové schéma Wienerova filtru

Wienerův filtr je invariantní filtr s konečnou impulsní charakteristikou. Základem není frekvenční odezva filtru, ale statické vlastnosti šumu. Dochází zde k restaurování signálu odstraněním šumu na základě statického popisu jeho vlastností.

Referenčním signálem je zde $d(n)$, chybový signál je $e(n)$. Vstupem filtru je signál $x(n)$ a výstupem $y(n)$. Tento výstup je generován jako lineární kombinace zpožděných vzorků výstupního signálu $y(n)$. Chyba $e(n)$ podmiňuje výběr koeficientů w_i a funkce, která tento výběr provádí by měla splňovat požadavek jednoduše vystopovatelného minima (resp. maxima), aby nedocházelo k nejednoznačnostem při konvergenci filtru. Tyto podmínky splňují právě FIR filtry s kritériem chyby ξ: [9]

$$\xi = E[|e(n)|^2], \tag{7}$$

kde $E[]$ vyjadřuje střední očekávanou hodnotu chyby. Tato funkce je nejjednodušší, má jednoduché minium a lze ji snadno vypočítat. Po dosazení chybového vektoru FIR filtru dostaneme:

$$\begin{aligned}
&\xi = E[|e(n)|^2] \\
&\xi = E[(d(n) - y(n))^2] \\
&\xi = E[d(n) - \mathbf{x}^T(n) \cdot \mathbf{w}) \cdot (d(n) - \mathbf{w}^T \cdot \mathbf{x}(n))] \\
&\xi = E[d^2(n) - d(n) \cdot (\mathbf{x}^T(n) \cdot \mathbf{w}) + (\mathbf{w}^T \cdot \mathbf{x}(n)) + (\mathbf{x}^T(n) \cdot \mathbf{w}) \cdot \mathbf{w}^T \cdot x(n)] \\
&\xi = E[d^2(n)] - E[d(n) \cdot \mathbf{x}^T(n) \cdot \mathbf{w}] - E[d(n)\mathbf{w}^T \cdot \mathbf{x}(n)] + E[\mathbf{x}^T(n) \cdot \mathbf{w} \cdot \mathbf{w}^T \cdot \mathbf{x}(n)] \\
&\xi = E[d^2(n)] - E[d(n) \cdot \mathbf{x}^T(n)] \cdot \mathbf{w} - \mathbf{w}^T \cdot E[d(n) \cdot x(n)] + \mathbf{w} \cdot E[\mathbf{x}^T(n) \cdot \mathbf{x}(n)] \cdot \mathbf{w}^T
\end{aligned} \tag{8}$$

.

Z Rov. 8 můžeme vytknout vektor koeficientů $\mathbf{w}$ z očekávané střední hodnoty E, jelikož $\mathbf{w}$ není statickou veličinou. Pro další úpravy zavedeme vektor $\mathbf{p}$.

$$\mathbf{p} = E[d(n)\mathbf{x}^T(n)]. \tag{9}$$

Vektor $\mathbf{p}$ má význam kroskorelační funkce a má rozměr $N-1$ prvků. Dále pak v Rov. 8 označíme výraz $E[\mathbf{x}^T(n) \cdot \mathbf{x}(n)]$ jako matici $\mathbf{R}$. Je to matice o rozměrech $(N-1)$x$(N-1)$ a má význam autokorelační funkce.

$$\mathbf{R} = \begin{matrix} r_{00} & r_{01} & r_{02} & \dots & r_{0,N-1} \\ r_{10} & r_{11} & r_{12} & \dots & r_{1,N-1} \\ r_{20} & r_{21} & r_{22} & \dots & r_{2,N-1} \\ \dots & \dots & \dots & \dots & \dots \\ r_{N-1,0} & r_{N-1,1} & r_{N-1,2} & \dots & r_{N-1,N-1} \end{matrix} \tag{10}$$

Po dosazení do Rov. 8 dostaneme:

$$\xi = E[d^2(n)] - 2\mathbf{p} \cdot \mathbf{w} + \mathbf{w} \cdot \mathbf{R} \cdot \mathbf{w}^T\,. \tag{11}$$

Funkce ξ je kvadratickou funkcí $\mathbf{w}$ a má jedno globální minimum. Polohu minima zjistíme pomocí parciálních derivací funkce ξ podle koeficientů $\mathbf{w}$ a položením zderivovaných funkcí $= 0$. S využitím vztahu platného pro symetrické matice:

$$\tfrac{d}{d\mathbf{w}}(\mathbf{A}\mathbf{w}) = \mathbf{A},\ \tfrac{d}{d\mathbf{w}}(\mathbf{w}M\mathbf{w}^T) = 2M\mathbf{w}, \tag{12}$$

kde $\mathbf{A}$ a M jsou matice. Hledaný vztah je tedy:

$$\nabla\xi = -2\mathbf{R}\mathbf{w} - 2\mathbf{p} = 0, \tag{13}$$

kde ∇ je operátor gradientu a odkud:

$$\mathbf{w_{opt}} \cdot \mathbf{R} = \mathbf{p}, \tag{14}$$

$$\mathbf{w_{opt}} = \mathbf{R^{-1}} \cdot \mathbf{p}. \tag{15}$$

Tato rovnice umožňuje vypočítat optimální váhy $\mathbf{w_{opt}}$ filtru a je známá jako Wiener-Hopfova rovnice.[12]

3.4 Charakteristika adaptivních algoritmů

Cílem algoritmů je vypočítat co nejlepší odhad koeficientů filtru, tak aby se výstupní signál co nejvíce podobal požadovanému signálu. V procesu nalezení koeficientů filtru se zpravidla řeší optimalizací chybové funkce. Ta může být obecně podmíněna stochasticky nebo deterministicky. Stochastický přístup vyžaduje větší množství měření ke spočtení statistiky. Tento postup využívají algoritmy LMS a jeho odvozeniny. Při deterministickém přístupu návrhu filtru vyžaduje výpočet charamtekistik z velkého počtu vzorků. Tento postup využívají alogirtmy RLS.

3.5 LMS algoritmus

Algoritmy LMS jsou ve své skupině těmi nejpoužívanějšími. Mají široké uplatnění hlavně díky své jednoduchosti a robustnosti. Tyto algoritmy jsou založeny na gradientním vyhledávacím algoritmu nazývaném také metoda největšího spádu. Závislost střední kvadratické odchylky výstupního chybového signálu adaptivního FIR filtru na koeficientech filtru je kvadratická křivka s jedním globálním minimem. Vyhledávání koeficientů s minimální střední kvadratickou odchylkou je založeno na posouvání koeficientů ve směru záporného gradientu křivky. Koeficienty jsou postupně přepočítávány, až je dosaženo bodu s nulovým gradientem.[10] Výstupní signál je vypočítán následovně:

$$\sum_{i=0}^{N-1} y(n) = \mathbf{w}^T \cdot \mathbf{x}(n), \tag{16}$$

kde $\mathbf{w}^T$ jsou koeficienty filtru a N je řád filtru.

Poté se vypočítá chybový signál:

$$e(n) = d(n) \cdot y(n). \tag{17}$$

Nakonec je proveden přepočet koeficientů filtru.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \nabla e^2(n), \tag{18}$$

kde μ je konvergenční konstanta, která určuje velikost kroku při pohybu směrem k minimu a tedy i rychlost konvergence.

Rovnici 18 lze rozepsat jako:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2 \cdot \mu \cdot e(n) \cdot \mathbf{x}(n-k), \tag{19}$$

kde k je délka filtru.

3.6 Normalizovaný LMS algoritums

Normalizovaný LMS (NLMS) je upravená forma standardního LMS algoritmu. Postup výpočtu je stejný jako u LMS algoritmu s tím rozdílem, že je konvergenční konstanta μ normalizována energií x, tedy:

$$\mu_{NLMS} = \frac{\mu}{||\mathbf{x}(n)||^2}. \tag{20}$$

To znamená, že při velkých hodnotách vstupu $x(n)$ bude postup k minimu pomalejší a naopak. Algoritmus NLMS tedy využívá vstupní hodnotu konvergenční konstanty pouze pro první iteraci. Poté je adaptivně přepočítávána.

3.7 RLS algoritmus

Algoritmy z rodiny RLS jsou typickými zástupci algoritmů založených na principu Kalmanovy filtrace [17]. V porovnání s algoritmy LMS mají algoritmy RLS větší konvergentní rychlost. To plyne z použití časového průměrování, které predikuje velmi přesné hodnoty. Problém těchto algoritmů je však v tom, že vyžadují komplikované matematické operace, z čehož plynou vyšší výpočetní a časové nároky než je tomu u LMS. Úlohy s použitím RLS algoritmu mají o řád vyšší složitost a jsou tudíž výrazně pomalejší.[5]. Pro časově neměnné signály RLS konverguje ke stejným koeficientům jako Wienerův filtr, pro časově proměnné signály RLS sleduje časové variace signálu.

Matematické odvození RLS algoritmu je k nalezení v některé z odborných publikací, např. [2], [16]. V této práci jsou popsány pouze kroky, nezbytné k implementaci algoritmu.

Výstupní signál je vypočítán následovně:

$$\sum_{i=0}^{N-1} y(n) = \mathbf{w}^T \cdot \mathbf{x}(n), \tag{21}$$

kde $\mathbf{w}^T$ jsou koeficienty filtru, $\mathbf{x}(n)$ je vektor vstupních vzorků a N je řád filtr. Poté je vypočten chybový signál:

$$e(n) = d(n) \cdot y(n). \tag{22}$$

Následuje přepočet koeficienty filtru podle rovnice:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{k}(n)e(n), \tag{23}$$

kde $\mathbf{k}(n)$ se nazývá ziskový vektor a určuje se podle vztahu:

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{\Psi}^{-1}(n-1)\mathbf{x}(n)}{1+\lambda^{-1}x^T\mathbf{\Psi}^{-1}(n-1)\mathbf{x}(n)}, \tag{24}$$

kde $\Psi(n)$ je inverzní autokorelační matice referenčního signálu $x(n)$ a λ je zapomínací faktor. V praxi se jeho hodnota voli mezi 0,98 a 1.[10] Pokud se rovná 1, znamená to, že se bude počítat se všemi vzorky. Inverzní autokorelační matice je adaptována následnovně:

$$\mathbf{\Psi}_\lambda^{-1}(n) = \lambda^{-1}(\mathbf{\Psi}_\lambda^{-1}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{\Psi}_\lambda^{-1}(n-1)). \tag{25}$$

Počáteční hodnota inverzní autokorelační matice:

$$\mathbf{\Psi}^{-1}(0) = \begin{matrix} \delta^{-1} & 0 & 0 & 0 \\ 0 & \delta^{-1} & 0 & 0 \\ 0 & 0 & \delta^{-1} & 0 \\ 0 & 0 & 0 & \delta^{-1} \end{matrix}, \tag{26}$$

kde δ je regularizační faktor.[5]

3.8 QR-RLS algoritmus

Odvozenina algoritmu RLS, která používá k přepočtu inverzní autokorelační matice tzv. QR rozklad (dekompozici).

QR rozklad je způsob, jak zapsat matici jako součin dvou matic, z nichž jedna je ortogonální, případně má alespoň vzájemně ortonormální sloupce, a druhá je v horním trojúhelníkovém tvaru. QR rozklad lze provést pomocí klasického nebo modifikovaného Gramova-Schmidtova algoritmu (případně s iteračním zpřesněním), nebo pomocí Householderových nebo Givensových transformačních matic. Při reálném výpočtu (tj. v aritmetice s konečnou přesností) se všechny zmíněné postupy výrazně liší v přesnosti a rychlosti výpočtu. Přesnost je klíčovým faktorem zejména v případě, že matice obsahuje lineárně závislé sloupce.

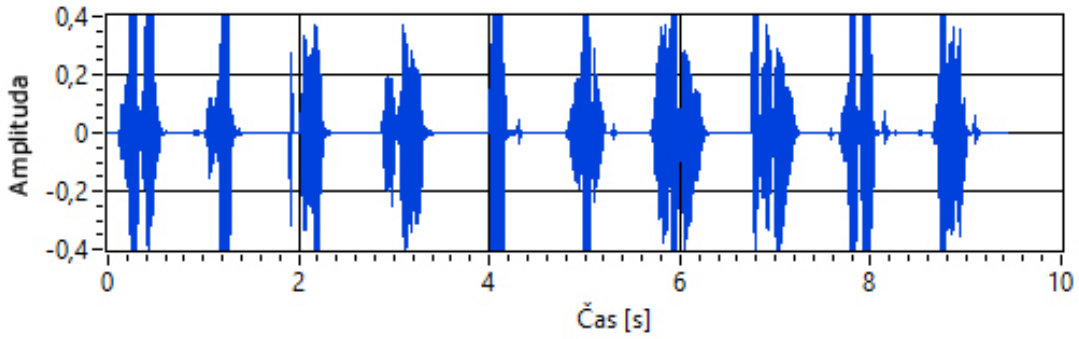
Postupy QR dekompozice jsou popsány v řadě odborných publikací, např. [13], [14], [15]. V rámci implementace algoritmu byla využita funkce pro QR dekompozice matice, intergrovaná ve vývojovém prostředí Microsoft Visual Studio. Postup výpočtu samotného signálu je obdobný jako u RLS algoritmu.

4 Analýza reálných aplikací

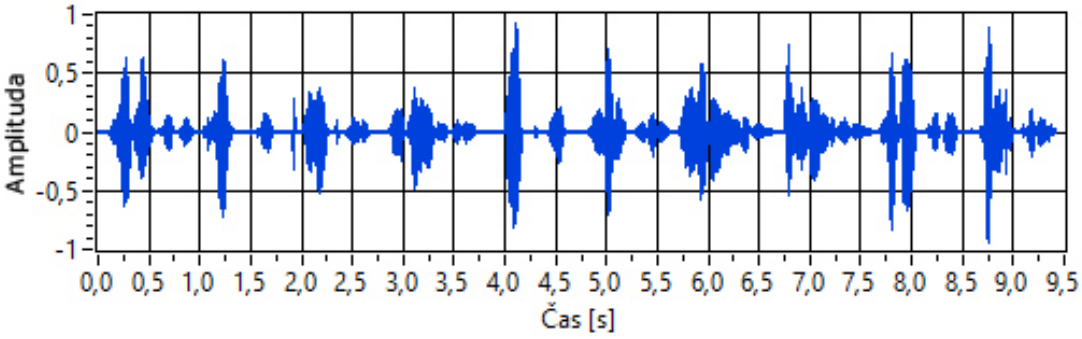
V této kapitole budou rozebrány reálné aplikace s využitím adaptivních filtrů. Základem každé z nich jsou vždy všechny čtyři implementované algoritmy, tedy LMS, NLMS, RLS a QR-RLS. Vybrané aplikace z níže uvedených budou vytvořeny s využitím LabVIEW v praktické části této práce.

4.1 Adaptivní potlačení echa

Úkolem potlačení echa je ideálně odstranit nežádoucí rušivou ozvěnu (echo) z komunikačních prostředků. Příkladem je situace, kdy jsou mikrofon a zdroj zvuku od sebe nedokonale izolovány. Tento jev se nazývá akustické echo a vzniká například při telefonním hovoru přes bezdrátovou sadu pro telefonování, kdy je tato sada nedokonale navržena.[6]

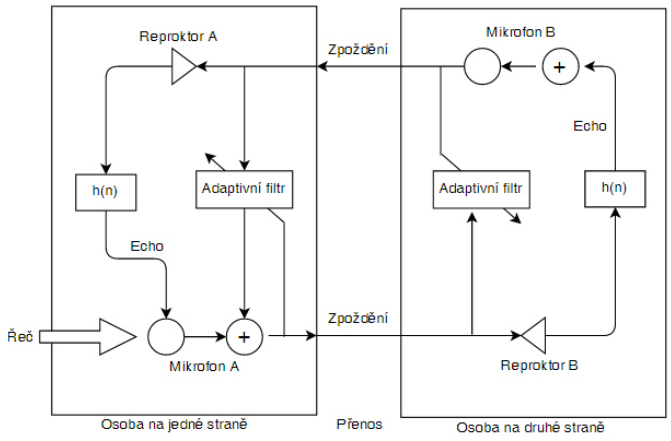


Obrázek 3: Časový průběh čisté nahrávky bez echa



Obrázek 4: Časový průběh nahrávky s přidaným echem

Echo může také pocházet z prostředí, kde se zvuk odráží od stěn a vrací zvukový signál zpět do přijímače [6]. Tato aplikace bude realizována a podrobněji rozebrána v praktické části práce.

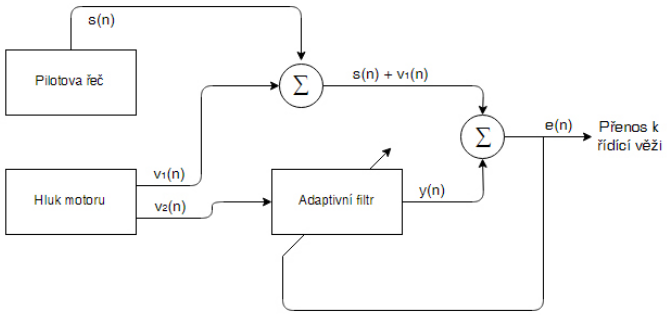


Obrázek 5: Schéma potlačení echa

4.2 Aktivní potlačení rušení

Dalším typickým využitím adaptivního filtru je potlačení rušení, kdy máme průběžně k dispozici doplňkovou informaci buď o užitečném signálu nebo o signálu rušivém. Typickým rušivým signálem je šum.[1]

Například pokud hluk z tryskového motoru letadla ruší komunikaci mezi pilotem a kontrolní věží. Tryskový motor dokáže vyvinout zvuk o hladině přes 140 dB, kdežto lidská řeč s pohybuje okolo 50 dB. K odstranění tohoto rušení se využívá právě adaptivní filtrace.[5]

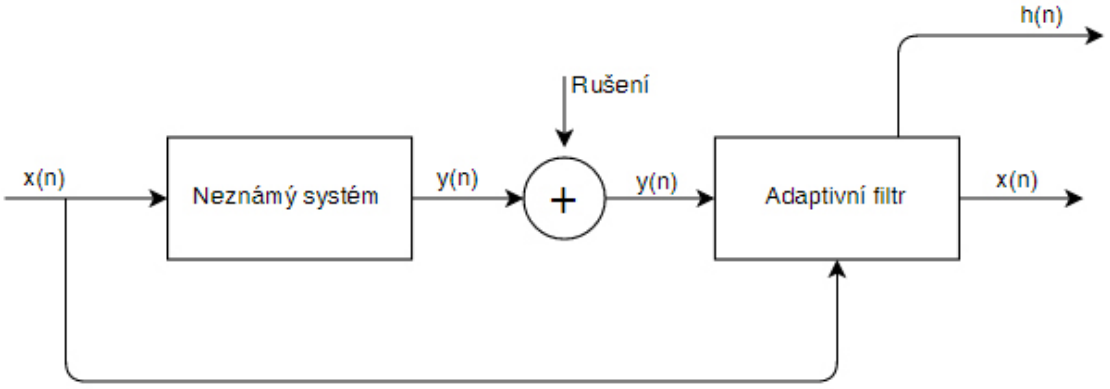


Obrázek 6: Schéma potlačení echa

- $s(n)$ - Pilotova řeč.
- $v_1(n)$ Hluk motoru přítomný spolu s $s(n)$ při přenosu vysílačkou.
- $v_2(n)$ Snímaný hluk motoru, který je využit jako doplňková informace o rušivém signálu.

4.3 Inverzní identifikace systému

Tato aplikace využívá kaskádového spojení neznámého systému a adaptivního filtru, které odpovídá tzv. inverzní identifikaci resp. modelování. Zde je adaptivní filtr vybuzen výstupním signálem neznámého systému, jehož vstupní signál je použitý jako referenční, takže výstupem adaptivního filtru je odhad originálního signálu.

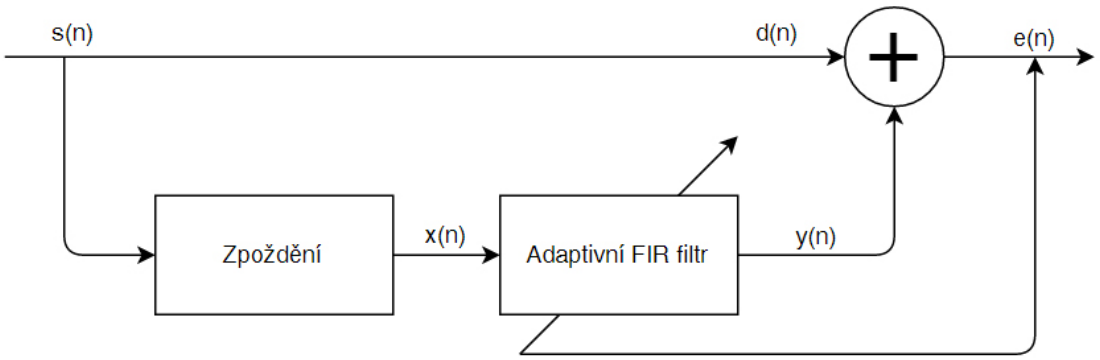


Obrázek 7: Inverzní identifikace neznáméno systému

Tato aplikace má praktické využití například při tzv. ekvalizaci kanálu. Ekvalizace kanálu je často využívána v telekomunikacích, kde neznámým systémem je sdělovací kanál s neznámou charakteristikou, jehož vlastnosti je třeba kompenzovat. Pro jistou dobu je zde k dispozici originální signál, který je použit jako referenční. Během této doby se filtr adaptuje a poté vyrovná vlastnosti kanálu.[1]

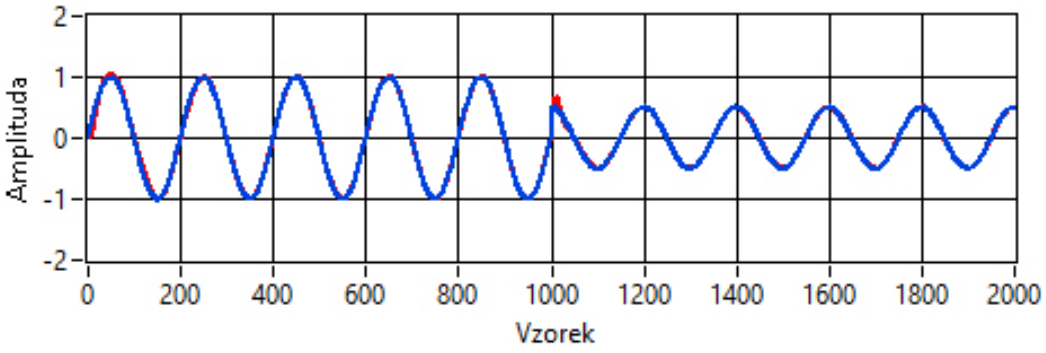
4.4 Adaptivní lineární predikce

Adaptivní lineární predikce využívá adaptivní filtr k odhadu budoucích hodnot signálu založeném na předchozích hodnotách signálu. Tuto predikci lze také využít pro kompresi obrazových signálů, jak je tomu například u metody lineárního predikčního kódování (LPC). Následující diagram znázorňuje blokový diagram adaptivní lineární predikce.[5]



Obrázek 8: Blokové schéma lineární predikce

- $s(n)$ - Signál posloupnost v čase n.
- $x(n)$ - Zpožděná verze $s(n)$.
- $y(n)$ - Výstup adaptivního filtru.
- $e(n)$ - Rozdíl $s(n)$ a $y(n)$.



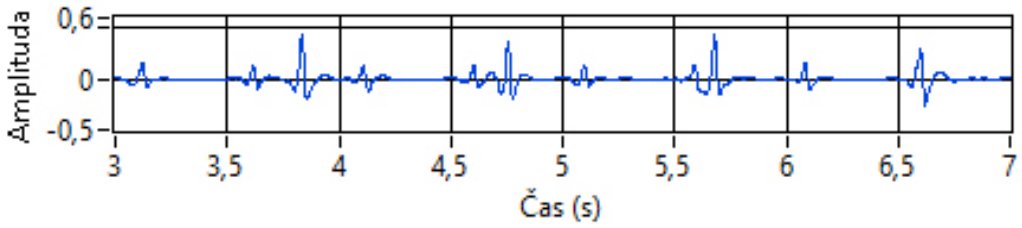
Obrázek 9: Ukázka predikce signálu

Na ukázce jsou vidět červená místa, což jsou úseky adaptace koeficientů filtru. Tato apliakce je realizována v praktické části.

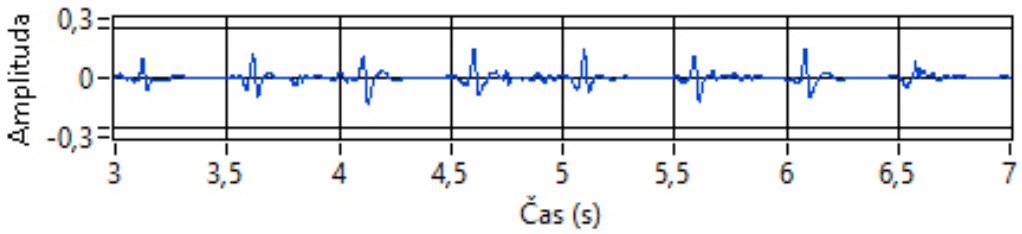
4.5 Extrakce EKG signálu plodu

Jak již bylo zmíněno, adaptivní filtrace se využívá také ve zdravotnictví. Například pro extrakci EKG signálu plodu v těle matky. Pro extrakci se využívá dvou EKG signálů z

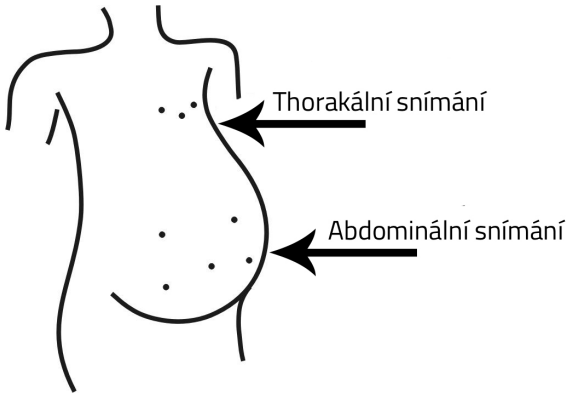
těla matky. První je pomocí elektrod snímám z hrudníku (tzv. torakální signál) a obsahuje pouze srdeční tlukot matky. Druhý (tzv. abdominální) signál je snímám z oblasti břicha a obsahuje tlukot matky i plodu uvnitř dělohy. Matčino EKG se pomocí adaptivních filtrů odfiltruje a výsledkem je EKG signál plodu.[5] Simulace této aplikace bude realizována v praktické části práce.



Obrázek 10: Průběh abdominálního EKG signálu matky



Obrázek 11: Vyfiltrovaný EKG signál plodu



Obrázek 12: Umístění elektrod na těle matky

5 LabVIEW Adaptive Filter Toolkit

Adaptive Filter Toolkit není součástí klasického LabVIEW a je nutno jej manuálně přidat jako doplněk. Součástí tohoto doplňku je také rozšíření nápovědy, kde jsou podrobně popsány možnosti a funkce toolkitu a teoretický základ o adaptivních filtrech. Nutono podotknout, že Adaptive Filter Toolkit v LabVIEW podporuje pouze filtry typu FIR.

Nápověda obsahuje přehledné rozdělení nabízených filtrů. Každá podstránka filtru obsahuje jednoduchý popis, blokový diagram s popisem jednotlivých vstupů a výstupů filtru. Často také obsahuje odkaz na virtuální přístroje jako příklad pro ilustraci a zároveň ukázku možnosti využití daného adaptivního filtru.

Ukázky rozhraní Adaptive Filter Toolkit jsou umístěny na přiloženém CD.

5.1 Funkce paletové nabídky Adaptive Fillter Toolkit

Popis jednotlivých funkcí, jejich vstupů a výstupů je vzhledem k rozsahu k dispozici na přiloženém CD.

5.2 Dynamicky linkovaná knihovna vybraných algoritmů

Vybrané adaptivní algoritmy LMS, NLMS, RLS a QR-RLS byly implementovány v prostředí Microsoft Visual Studio v jazyce C#. Všechny tyto algoritmy byly poté importovány prostřednictvím DLL knihovny do programu LabVIEW. Vzorový příklad importu DLL knihovny je k dispozici na přiloženém CD.[11].

6 Praktická část

V praktické části byly vybrány a realizovány tři aplikace:

- Adaptivní potlačení echa.
- Adaptivní lineární predikce.
- Extrakce EKG signálu plodu.

6.1 Implementace algoritmů v jazyce C#

V rámci praktické části práce byly implementovány do DLL knihovny algoritmy LMS, NLMS, RLS a QR-RLS. Pro porovnání byly stejné algoritmy vybrány také z řad Adaptive Filter Toolkit. Projekt s kompletními kódy naprogramovaných algoritmlů je umístěn na přiloženém CD.

6.1.1 Implementace LMS algoritmu

Hlavička funkce:

```
public void lmsFilter(double[] x, double[] d, double mu, int N, out double[] y, out double[] e)
```

Vstupní parametry:

- x - Vstupní signál.
- d - Referenční signál.
- mu - Konvergenční kostanta μ.
- N - Délka filtru.

Výstupní parametry:

- e - Chybový signál.
- y - Výstupní signál.

Načtení délky vstupního signálu a zadefonování lokálních proměnných pro výpočet. Délka vstupního signálu M je při deklaraci nastavena také výstupním signálům e a y. Proměnná **w** představuje vektor koeficientů filtru a má délku N, což je délka filtru.

```
int M = x.Length;
double[] w = new double[N];
double[] y_ = new double[M];
double[] e_ = new double[M];
double sum;
```

Následující úsek probíhá v **for** cyklu, s počtem iterací M.

```
for (int t = N; t < M; t++)
```

V první části každé iterace probíhá samotná filtrace a výpočet chyby e.

```
sum = 0;
for (int j = (t − 1); j >= (t − N); j−−){
    sum += (w[t − j − 1] ∗ x[j]) ;
}
y_[t − 1] = sum;
e_[t − 1] = d[t − 1] − y_[t − 1];
```

Nyní proběhne přepočet koeficienů filtru $\mathbf{w}$.

```
for (int j = (t − 1); j >= (t − N); j−−){
   w[t − j − 1] = w[t − j − 1] + (mu ∗ e_[t − 1] ∗ x[j]) ;
}
```

Pokračuje se další iterací do doby, než je vyčerpána celá délka vstupního signálu. Posledním krokem je uložení výsledných hodnot do výstupních parametrů funkce.

```
y = y_;
e = e_;
```

6.1.2 Implementace NLMS algoritmu

Hlavička funkce:

```
public void nlmsFilter(double[] x, double[] d, double mu, int N, out double[] y, out double[] e)
```

Vstupní parametry:

- x - Vstupní signál.
- d - Referenční signál.
- mu - Konvergenční kostanta μ.
- N - Délka filtru.

Výstupní parametry:

- e - Chybový signál.
- y - Výstupní signál.

Načtení délky vstupního signálu a zadefonování lokálních proměnných pro výpočet. Délka vstupního signálu M je při deklaraci nastavena také výstupním signálům e a y. Proměnná $\mathbf{w}$ představuje vektor koeficientů filtru a má délku N, což je délka filtru.

```
int M = x.Length;
double[] w = new double[N];
double[] y_ = new double[M];
double[] e_ = new double[M];
double sum;
```

Následující úsek probíhá v **for** cyklu, s počtem iterací M.

```
for (int t = N; t < M; t++)
```

V první části každé iterace probíhá samotná filtrace a výpočet chyby e a přepočet konvergenční konstanty.

```
sum = 0;
for (int j = (t − 1); j >= (t − N); j−−){
    sum += (w[t − j − 1] ∗ x[j]) ;
    mu += (x[j] ∗ x[j]) ;
}
y_[t − 1] = sum;
e_[t − 1] = d[t − 1] − y_[t − 1];
mu = 1 / mu;
```

Nyní proběhne přepočet koeficienů filtru $\mathbf{w}$.

```
for (int j = (t − 1); j >= (t − N); j−−){
   w[t − j − 1] = w[t − j − 1] + (mu ∗ e_[t − 1] ∗ x[j]) ;
}
```

Pokračuje se další iterací do doby, než je vyčerpána celá délka vstupního signálu. Posledním krokem je uložení výsledných hodnot do výstupních parametrů funkce.

```
y = y_;
e = e_;
```

6.1.3 Implementace RLS algoritmu

Hlavička funkce:

```
public void rlsFilter (double[] x, double[] d, int N, double lambda, double delta, out double[] y,
    out double[] e)
```

Vstupní parametry:

- x - Vstupní signál.
- d - Referenční signál.
- $lambda$ - Zapomínací faktor λ.
- $delta$ - Regularizační faktor δ.
- N - Délka filtru.

Výstupní parametry:

- e - Chybový signál.
- y - Výstupní signál.

Načtení délky vstupního signálu a zadefonování lokálních proměnných pro výpočet. Délka vstupního signálu M je při deklaraci nastavena také výstupním signálům e a y. Zadefinování vektoru koeficientů $\mathbf{w}$, ziskového vektoru $\mathbf{g}$, vektoru vzorků vstupního signálu $\mathbf{X1}$ a autokorelační matice $\mathbf{P}$.

```
int M = x.Length;
Vector<double> w = Vector<double>.Build.Dense(N);
Vector<double> g = Vector<double>.Build.Dense(N);
Vector<double> X1 = Vector<double>.Build.Dense(N);
Matrix<double> P = Matrix<double>.Build.DenseDiagonal(N,N, 1/delta);
double[] y_ = new double[M];
double[] e_ = new double[M];
```

Následující úsek probíhá v **for** cyklu, s počtem iterací M.

```
for (int t = N; t < M; t++)
```

V první části každé iterace probíhá samotná filtrace a výpočet chyby e.

```
double sum = 0;
for (int j = (t - 1); j >= (t - N); j--){
    X1[t - j - 1] = x[j];
            sum += (w[t - j - 1] * x[j]);
}
 y_[t - 1] = sum;
e_[t - 1] = d[t - 1] - y_[t - 1];
```

Nyní proběhne přepočet ziskového vektoru $\mathbf{g}$, matice $\mathbf{P}$ a koeficienů filtru $\mathbf{w}$.

```
g = (P * X1) / (lambda * P.Multiply(X1) * X1);
P = (1 / lambda) * P - (1 / lambda) * g * P.Multiply(X1);
for (int j = (t - 1); j >= (t - N); j--){
   w[t - j - 1] = w[t - j - 1] + (g[t - j - 1] * e_[t - 1]);
}
```

Pokračuje se další iterací do doby, než je vyčerpána celá délka vstupního signálu. Posledním krokem je uložení výsledných hodnot do výstupních parametrů funkce.

```
y = y_;
e = e_;
```

6.1.4 Implementace QR-RLS algoritmu

Hlavička funkce:

```
public void rlsFilter (double[] x, double[] d, int N, double lambda, double delta, out double[] y,
    out double[] e)
```

Vstupní parametry:

- x - Vstupní signál.
- d - Referenční signál.
- $lambda$ - Zapomínací faktor λ.
- $delta$ - Regularizační faktor δ.
- N - Délka filtru.

Výstupní parametry:

- e - Chybový signál.
- y - Výstupní signál.

Načtení délky vstupního signálu a zadefonování lokálních proměnných pro výpočet. Délka vstupního signálu M je při deklaraci nastavena také výstupním signálům e a y. Zadefinování vektoru koeficientů $\mathbf{w}$, ziskového vektoru $\mathbf{g}$, vektoru vzorků vstupního signálu $\mathbf{X1}$ a autokorelační matice $\mathbf{P}$.

```
int M = x.Length;
Vector<double> w = Vector<double>.Build.Dense(N);
Vector<double> g = Vector<double>.Build.Dense(N);
Vector<double> X1 = Vector<double>.Build.Dense(N);
Matrix<double> P = Matrix<double>.Build.DenseDiagonal(N,N, 1/delta);
double[] y_ = new double[M];
double[] e_ = new double[M];
```

Následující úsek probíhá v **for** cyklu, s počtem iterací M.

```
for (int t = N; t < M; t++)
```

V první části každé iterace probíhá samotná filtrace a výpočet chyby e.

```
double sum = 0;
for (int j = (t - 1); j >= (t - N); j--){
    X1[t - j - 1] = x[j];
            sum += (w[t - j - 1] * x[j]);
}
 y_[t - 1] = sum;
e_[t - 1] = d[t - 1] - y_[t - 1];
```

Nyní proběhne přepočet ziskového vektoru $\mathbf{g}$, matice $\mathbf{P}$ pomocí QR dekompozice a koeficienů filtru $\mathbf{w}$.

```
g = (P * X1) / (lambda * P.Multiply(X1) * X1);
P.QR();
for (int j = (t - 1); j >= (t - N); j--){
   w[t - j - 1] = w[t - j - 1] + (g[t - j - 1] * e_[t - 1]);
}
```

Pokračuje se další iterací do doby, než je vyčerpána celá délka vstupního signálu. Posledním krokem je uložení výsledných hodnot do výstupních parametů funkce.

```
y = y_;
e = e_;
```

6.2 Úloha 1: Adaptivní potlačení echa

6.2.1 Prvky čelního panelu

Čelní panel obsahuje volbu algoritmu s možností zvolení LabVIEW nebo C# implementace. Dále umožňuje volbu cesty ke zvukové nahrávce pro testování. Je však třeba zachovat formát zvukové stopy, tedy .WAV. Pro ilustraci je zde nahrávka již vložena. Pro účely testování je zde zobrazen také čas filtrace. Lze tedy porovnávat rychlost jednotlivých algoritmů. Dále pomocí posuvných jezdců můžeme konfigurovat:

- Zpoždění echa.
- Délku filtru.
- Konvergenční konstantu (pro LMS a NLMS).
- Zapomínací faktor (pro RLS a QR-RLS).
- Regularizační faktor (pro RLS a QR-RLS).

V ovládacím bloku je také tlačítko STOP pro zastavení aplikace.

Na právé straně je pak umístěno zobrazení čisté zvukové stopy, stopy s přidaným echem a výsledné stopy po filtraci. Vedle každého grafu je tlačítko pro přehrání ukázky.

Podoba čelního panelu a blokové schéma aplikace jsou k dispozici v příloze této práce.

6.2.2 Základní struktura aplikace

Všechny součásti blokového schématu jsou uzavřeny ve smyčce While Loop. Tato struktura zajišťuje opakování algoritmu po celou dobu spuštění programu. Iteraci lze zastavit kliknutím na tlačítko STOP. Nejprve je k čisté nahrávce přidáno echo. Tento signál je spolu s čistým signálem a potřebnými parametry přiveden do zvoleného adaptivního filtru, který signály zpracuje. Výsledkem filtrace je signál, který je poté zobrazen v průběhu, kde je možnost jej přehrát.

6.2.3 Výsledky experimentů

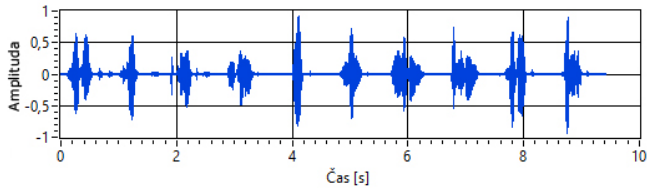
Navržená aplikace byla ostestována na vlastní řečové nahrávce o délce cca 10 sekund (420 000 vzorků). Během testování se obecně algoritmy implementované v jazyce C# prokázaly jako pomalejší. Algoritmy z rodiny RLS implementované v C# trvaly dokonce několikanásobně déle než jejich ekvivalenty z LabVIEW (cca 4 minuty). V Tabulce 1 je uvedena rychlost konvergence algoritmů z rodiny LMS pro pět různých nastavení konvergenční konstanty, při délce filtru $N = 100$.

Tabulka 1: Úloha 1: Rychlost konvergence LMS a NLMS algoritmů

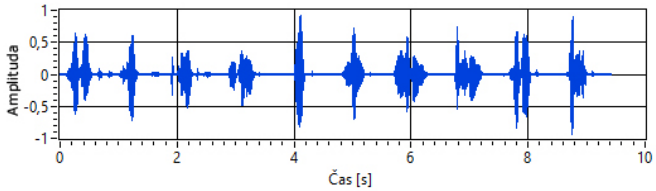
Rychlost konvergence [vzorky]				
μ	LMS	NLMS	C# LMS	C# NLMS
0,00025	-	-	-	-
0,0005	-	390 000	-	380 000
0,001	370 000	330 000	370 000	280 000
0,003	300 000	130 000	330 000	110 000
0,01	150 000	60 000	160 000	60 000

Z Tabulky 1 je patrné, že rychlost konvergence obou metod je velice podobná.

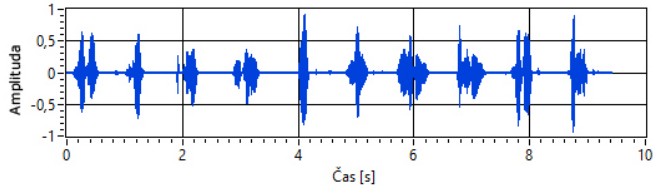
V případě algoritmů RLS a QR-RLS byla pro účely testování zvolena hodnota zapomínacího faktoru $\lambda = 1$, tedy nekonečná paměť. Regularizační fator δ byl zvolen 0,00001, což je doporučená výchozí hodnota nastavení v Adaptive Filter Toolkit. Následující obrázky zobrazují průběhy filtrace obou metod implementace.



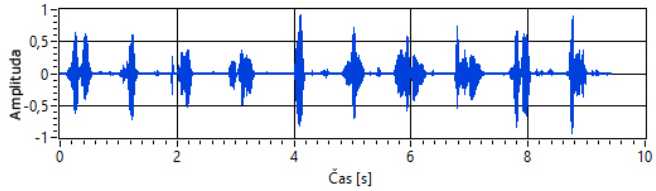
Obrázek 13: Průběh filtrace rodiny LMS algoritmů z LabVIEW



Obrázek 14: Průběh filtrace rodiny LMS algoritmů implementovaných v C#



Obrázek 15: Průběh filtrace rodiny RLS algoritmů z LabVIEW



Obrázek 16: Průběh filtrace rodiny RLS algoritmů implementovaných v C#

Z obrázků je patrné, že konvergenční rychlost obou metod je velice nízká, avšak doba výpočtu RLS a QR-RLS algoritmů implementovaných v jazyce C# je nesrovnatelně vyšší. Zatím co výpočet algoritmů z LabVIEW trval vždy zhruba 70 sekund, agoritmům v C#

trval výpočet téměř 8 minut. Tento rozdíl je způsobem množstvím iterací, které se provádí uvnitř implementovaného algoritmu a neoptimalizovaným kompilováním C# kódu z DLL knihoven do prostředí LabVIEW. V Tabulce 2 jsou uvedeny časy výpočtu jednotlivých algoritmů obou metod, při nastavení $\mu = 0,001$, $\lambda = 1$, $delta = 0,00001$ a $N = 100$. Tyto hodnoty byly testováním určeny jako optimální z hlediska rychlosti konvergence a stability filtru.

Tabulka 2: Úloha 1: Porovnání časů výpočtů jednotlivých algoritmů

Čas výpočtu algoritmu [s]	
LMS	0,48
NLMS	0,56
RLS	35,42
QR-RLS	42,73
C# LMS	1,18
C# NLMS	1,51
C# RLS	442,28
C# QR-RLS	457,37

U filtrace řeči je nejdůležitější kvalita výsledného signálu. Tu lze objektivně vyjádřit pomocí tzv. SNR, což je poměr odstupu signálu od šumu, vyjadřovaný v decibelech. Tento poměr vyjadřuje, kolikrát je výkon signálu větší, než výkon šumu, jímž je signál zkreslen. Poměr SNR je definován podle vztahu[8]:

$$SNR = 10log_{10}\frac{P_s}{P_n}, \tag{27}$$

kde P_s a P_n označují výkon signálu. Je-li hodnota SNR=0 dB znamená to, že signál i šum mají stejný výkon. Při SNR > 0 je výkon signálu větší, než šumu a při SNR < 0 je tomu naopak.[8]. Pro usnadnění výpočtu SNR byla naprogramovaná funkce v jazyce C# a importována do prostředí LabVIEW pomocí DLL knihovny, podobně jako implementované algoritmy.

V Tabulce 3 jsou zaznamenány SNR všech zkoumaných algoritmů. Hodnota SNR_{Sig} je pro vstupní signál před filtrací, hodnota SNR_{Fil} je vypočtena po filtraci.

Tabulka 3: Úloha 1: Porovnání SNR jednotlivých algoritmů

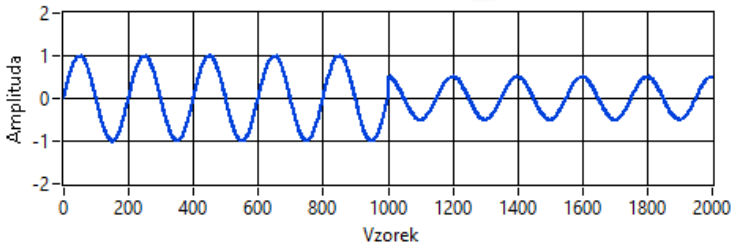
Algoritmus	$\text{SNR}_{Sig}[dB]$	$\text{SNR}_{Fil}[dB]$
LMS	14,62	29,9
NLMS	14,62	36,3
RLS	14,62	38,5
QR-RLS	14,62	36,6
C# LMS	14,62	31,2
C# NLMS	14,62	39,9
C# RLS	14,62	41,2
C# QR-RLS	14,62	40,8

Z výsledků je patrné, že vyfiltrované signály mají větší SNR než původní signál, tzn. jejich výkon oproti výkonu šumu byl zvýšen. Lepších výsledků dosáhly algoritmy typu RLS. Pokud budeme porovnávat LabVIEW a implementované algoritmy, nepatrně lepších výsledků dosáhly algoritmy implementované v jazyce C#.

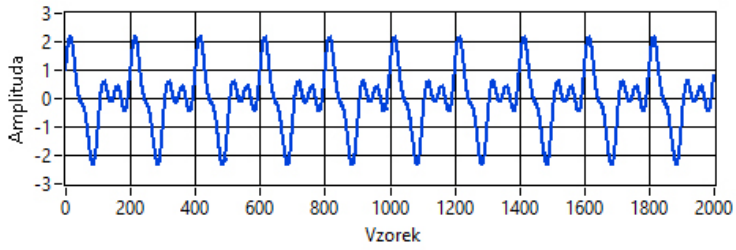
6.3 Úloha 2: Adaptivní lineární predikce

6.3.1 Prvky čelního panelu

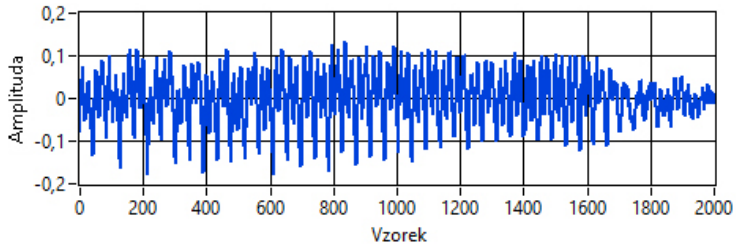
Čelní panel této aplikace také obsahuje možnosti konfigurace typu vstupního signálu. Volit lze mezi signálem Sin + Cos, Tóny nebo Řečovým signálem.



Obrázek 17: Sin + Cos vstupní signál



Obrázek 18: Vstupní signál s tóny



Obrázek 19: Řečový vstupní signál

Na čelním panelu této aplikace naleznete volbu algoritmu s možností zvolení LabVIEW nebo C# implementace. Pro účely testování je zde zobrazen také čas filtrace. Lze tedy porovnávat rychlost jednotlivých algoritmů. Dále pomocí posuvných jezdců můžeme konfigurovat:

- Délku filtru.
- Konverenční konstantu (pro LMS a NLMS).
- Zapomínací faktor (pro RLS a QR-RLS).
- Regularizační faktor (pro RLS a QR-RLS).

Na pravé straně jsou zobrazeny křivky vstupního, předpovídaného signálu a chyba předpovědi.

Podoba čelního panelu a blokové schéma aplikace jsou k dispozici v příloze této práce.

6.3.2 Základní struktura aplikace

Vybraný signál je spolu s potřebnými parametry přiveden do zvoleného adaptivního filtru, který signály zpracuje. Chybový signál je zobrazen odděleně. Před zobrazením je ještě odečten od původního vstupního signálu a ten je poté zobrazen jako předpovídaný signál.

6.3.3 Výsledky experimentů

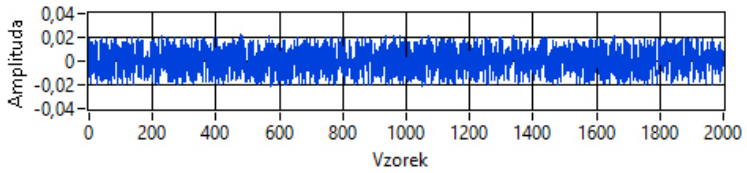
Navržená aplikace byla ostestována na třech typech signálů, konkrétně Sin + Cos, tóny a řeč o délce 2000 vzorků. Hodnotícími faktory u této aplikace byla rychlost kovergence, celkový čas výpočtu a chyba přesnosti. Všechny tyto faktory byly testovány při nastavení filtrů $\mu = 0,001$, $\lambda = 1$, $delta = 0,00001$ a $N = 100$. V Tabulce 4 můžeme vidět rychlost konvergence a celkový čas výpočtu algoritmů. Uvedené hodnoty jsou pro řečový signál.

Tabulka 4: Úloha 2: Porovnání rychlosti konvergence a časů výpočtů jednotlivých algoritmů

Algoritmus	Rychlost konvergence [vzorky]	Čas výpočtu [s]
LMS	110	0,03
NLMS	90	0,05
RLS	180	0,32
QR-RLS	170	0,18
C# LMS	100	0,08
C# NLMS	90	0,10
C# RLS	230	6,64
C# QR-RLS	190	7,12

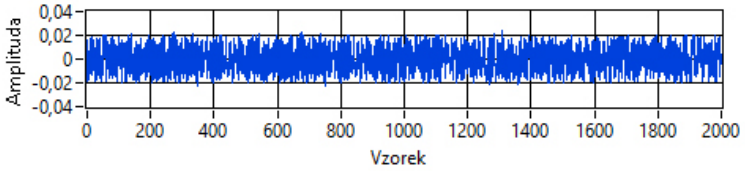
Rychlost konvergence je zhruba podobná v případě obou metod. Časy výpočtu se však opět liší. Nutno podotknout, že první iterace algoritmů implementovaných v jazyce C# byla vždy pomalejší a to z důvodů načítání DLL knihovny do paměti a kompilace C# na úroveň LabVIEW.

Chyba přesnosti byla obecně větší u algoritmu rodiny LMS a to v případě obou metod. Průběh LMS a NLMS byl takřka stejný, uveden je tedy pouze jeden. Algortimy LMS a NLMS implementované v C# měly nepatrně větší chybu přesnosti než jejich ekvivalenty z LabVIEW.

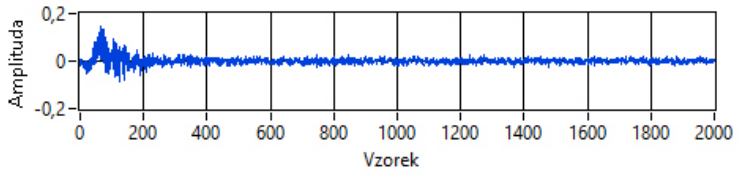


Obrázek 20: Chyba přesnosti rodiny LMS algoritmů (použitý LMS) z LabVIEW

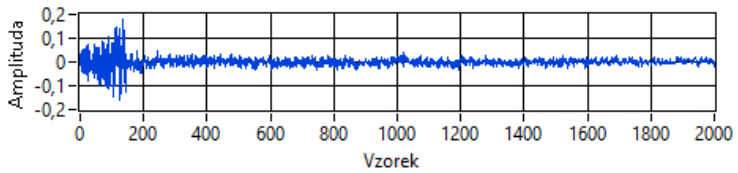
Algoritmy RLS a QR-RLS vykazovaly pouze větší chybovost na začátku filtrace, poté byla výrazně nižší než u rodiny LMS. U obou porovnávaných metod byly velice podobné.



Obrázek 21: Chyba přesnosti rodiny LMS algoritmů (použitý) algoritmů C#



Obrázek 22: Chyba přesnosti rodiny algoritmů RLS (použitý RLS) z LabVIEW



Obrázek 23: Chyba přesnosti rodiny algoritmů RLS (použitý RLS) algoritmů C#

6.4 Úloha 3: Extrakce EKG signálu plodu

6.4.1 Prvky čelního panelu

Čelní panel obsahuje, stejně jako předchozí aplikace, volbu algoritmu s možností zvolení LabVIEW nebo C# implementace. Pro ilustraci je zde zobrazen také čas filtrace. Lze tedy porovnávat rychlost jednotlivých algoritmů. Dale pomocí posuvných jezdců můžeme konfigurovat:

- Délku filtru.
- Konverenční konstantu (pro LMS a NLMS).
- Zapomínací faktor (pro RLS a QR-RLS).
- Regularizační faktor (pro RLS a QR-RLS).

Na právé straně jsou zobrazeny křivky torakálního EKG signálu matky, abdominálního EKG signálu matky a odfiltrovaného EKG signálu plodu

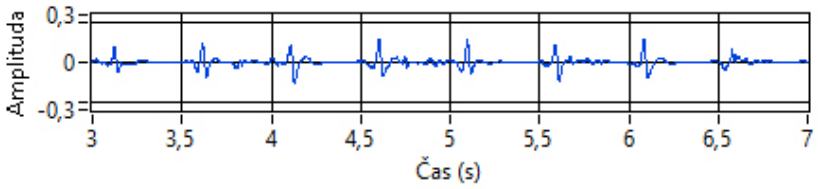
Podoba čelního panelu a blokové schéma aplikace jsou k dispozici v příloze této práce.

6.4.2 Základní struktura aplikace

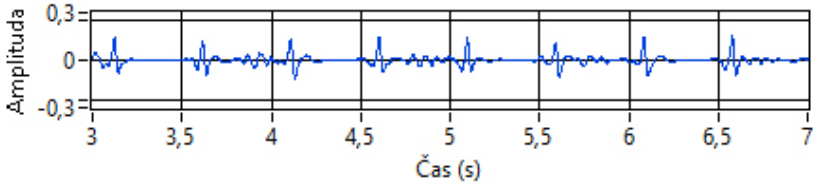
Oba signály matky jsou spolu s potřebnými parametry přivedeny do zvoleného adaptivního filtru, který signály zpracuje. Chybový signál (EKG signál plodu) je zobrazen v samostatném grafu.

6.4.3 Výsledky experimentů

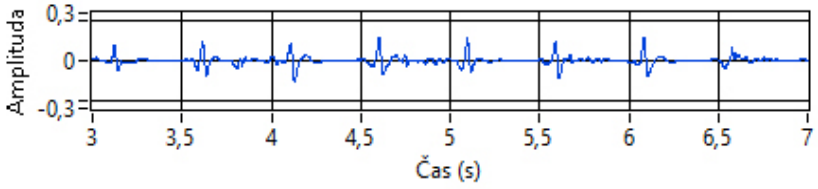
V této úloze byly testované algoritmy opět srovnávány podle hodnoty odstupu signálu od šumu SNR a z hlediska rychlosti výpočtu. Testování proběhlo na cca 7 sekundových simulovaných EKG signálech, které představovaly signál z hrudníku matky a signál z břicha matky (společně se signálem plodu). Podstatou aplikace je vyfiltrovat matčin EKG signál ze společného signálu a získat tak čistý EKG signál plodu. Tento úkol zvládly všechny algoritmy v relativně nízkém čase. RLS a QR-RLS algoritmy s rychlejší konvergencí ke správnému výsledku.



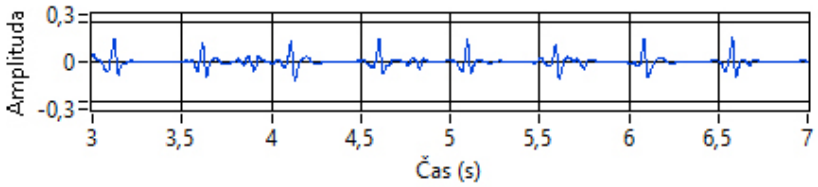
Obrázek 24: Výsledný EKG signál při použití algoritmů rodiny LMS (použitý LMS) z LabVIEW



Obrázek 25: Výsledný EKG signál při použití algoritmů rodiny RLS (použitý RLS) z LabVIEW



Obrázek 26: Výsledný EKG signál při použití algoritmů rodiny LMS (použitý LMS) v C#



Obrázek 27: Výsledný EKG signál při použití algoritmů rodiny LRS (použitý RLS) v C#

V Tabulce 5 jsou vidět časy výpočtů jednotlivých algoritmů, opět se implementované algoritmy ukázaly jako pomalejší. Testování proběhlo při nastavení filtrů $\mu = 0,01$, $\lambda = 1$, $delta = 0,00001$ a $N = 60$. V Tabulce 6 jsou uvedeny výsledky porovnávání SNR jednotlivých algoritmů. V této úloze byl EKG signál matčina srdce brán jako šum, který zkresloval požadovaný signál plodu. V původním signálu jsou tedy záporné hodnoty SNR, protože je výkon EKG signálu matky výrazně silnější.

Tabulka 5: Úloha 3: Porovnání rychlosti výpočtů jednotlivých algoritmů

Algoritmus	Čas výpočtu [s]
LMS	0,11
NLMS	0,12
RLS	0,68
QR-RLS	0,59
C# LMS	0,21
C# NLMS	0,25
C# RLS	5,49
C# QR-RLS	6,68

Tabulka 6: Úloha 3: Porovnání SNR jednotlivých algoritmů

Algoritmus	$\text{SNR}_{Sig}[dB]$	$\text{SNR}_{Fil}[dB]$
LMS	-2,01	5,26
NLMS	-2,01	6,33
RLS	-2,01	8,27
QR-RLS	-2,01	8,32
C# LMS	-2,01	3,87
C# NLMS	-2,01	4,46
C# RLS	-2,01	9,08
C# QR-RLS	-2,01	9,15

Z výsledků je opět patrná lepší kvalita filtrace RLS a QR-RLS algoritmů v případě obou metod implementace. Algoritmy implementované v jazyce C# vykazovaly zanedbatelně lepší výsledky než jejich ekvivalenty z LabVIEW.

7 Diskuze výsledků

Z experimentů všech tří uloh vyplynula lepší rychlost kovergence algoritmů RLS a QR-RLS a to jak těch z LabVIEW, tak těch implementovaných v jazyce C#. Algoritmy z LabVIEW měly téměř ve všech případech kratší dobu výpočtu, v případě RLS algortimů v první aplikaci byl skok i v řádech minut. Algoritmy implementované v jazyce C# byly obecně pomalejší a to z důvodů nedokonalého a neoptimalizovaného kompilátoru DLL knihoven z jazyka C# na úroveň aplikace LabVIEW, který vždy prodloužil dobu výpočtu. U algoritmů RLS je to pak také velké množství vnořených cyklů, které probíhaly při přepočítávání inverzní autokorelační matice, ziskového vektoru a vektoru koeficientů. Například v úloze č. 2 byl čas výpočtu algoritmů z DLL při spuštění jedné iterace delší zhruba o 1 sekundu. Po spuštění aplikace v kontinuální režimu se čas výpočtu v dalších iteracích ustálil na čas podobný integrovaným algoritmům. Tato zajímavá skutečnost vypovídá o tom, že určitou část výpočtu zabírá načítání DLL knihovny do paměti.

Na druhou stranu pokud algoritmy porovnáme z pohledu vypočtených hodnot SNR, můžeme v tomto případě algoritmy implementované v jazyce C# označit za lepší. Výsledné SNR těchto algoritmů bylo ve většině případů o několik jednotek decibelů větší. Výsledný signál tedy lze objektivně hodnotit jako kvalitnější. Při porovnání SNR algoritmů z rodiny LMS a RLS bylo zjištěno, že RLS algoritmy vykazují větší odstup výsledného signálu od šumu. To je způsobeno hlavně menší rychlostí konvergence ke správnému výsledku algoritmů LMS a NLMS.

Další oblastí porovnávání byla výpočetní náročnost algoritmů, která byla jednoznačně vyšší u algoritmů typu RLS a QR-RLS. V případě C# implementace byla ještě zvýšena nutností kompilace kódu na úroveň LabVIEW.

Celkově bylo tedy zjištěno, že RLS a QR-RLS algoritmy vykazují lepší filtrační vlastnosti, což je ovšem na úkor vyšší výpočetní náročnosti těchto algoritmů. Větší výpočetní náročnost tedy nutně znamená vyšší náklady na realizaci systémů pro samotnou adaptivní filtraci, která roste s potřebnými výpočetními nároky.

8 Závěr

Výsledkem této práce jsou naimplementované adaptivní algoritmy LMS, NLMS, RLS a QR-RLS v jazyce C#, které je možnost využít v program LabVIEW prostřednictvím DLL knihovny. Z algoritmů je vytvořen tzv. Toolkit, který rozšiřuje paletovou nabídku funkcí zmíněného programu. V další fázi práce byly vytvořeny vzorové aplikace pro otestování naimplementovaných algoritmů a srovnání s řešením Adaptive Filter Toolkit. Ve všech aplikacích je proto možné zvolit mezi implementovanými algoritmy a algoritmy z prostředí LabVIEW.

V aplikacích byly otestovány všechny implementované algoritmy a byly zjištěny výhody a nevýhody jednotlivých typů. Zkoumané adaptivní algoritmy typu LMS a NLMS jsou jednodušší a mají menší nároky na výpočetní výkon. Mají však výrazně menší rychlost konvergence ke správnému výsledku. Druhou skupinou testovaných byly algoritmy RLS a QR-RLS, které jsou naopak složitější a pro výpočet vyžadují mnohem náročnější matematické operace. Tento nedostatek však dokáží kompenzovat výbornou rychlostí konvergence a větší přesností. Algoritmy RLS a QR-RLS obstály stejně dobře v porovnání SNR, tedy poměr signálu k šumu. Celkově algoritmy RLS a QR-RLS vykazují lepší filtrační vlastnosti. Zároveň je však nutné podotknout, že vzhledem k matematické náročnosti těchto algoritmů zcela jistě vzrostou náklady na realizaci potřebného výkonnějšího filtračního systému. Na druhou stranu, vzhledem k rostoucímu výkonu hardwarových zařízení, lze využívat i komplikovanější adaptivní algoritmy k řešení rozsáhlejších úloh za relativně nízkou cenu. Adaptivní filtrace má dnes velké možnosti využití, vzhledem k rostoucí hardwarové výkonnosti počítačů, signálových zařízení a mobilních zařízení. Lze tedy využívat i komplikovanější adaptivní algoritmy v rozsáhlejších úlohách.

Všechny vytvořené algoritmy a aplikace lze využít k demonstraci adaptivních filtrů a k výukovým účelům. Pro tento účel byly vytvořeny dokumenty podrobně popisující realizované aplikace a manuály.

Vzhledem k velkému potenciálu adaptivních filtrů v mnoha oborech by bylo vhodné rozšíření LabVIEW Adaptive Filter Toolkit o další algoritmy. Současná nabídka je velice omezená, obsahuje prakticky jen čtyři zmíněné algoritmy. Vhodné by bylo také optimalizovat kompilování DLL knihoven tak, aby nedocházelo ke zbytečným ztrátám v časech výpočtů.

9 Reference

[1] JAN, Jiří. Číslicová filtrace, analýza a restaurace signálů. 2. upr. a rozš. vyd. Brno: VUTIUM, 2002. ISBN 80-214-1558-4.

[2] FARHANG-BOROUJENY, B. *Adaptive filters: theory and applications*. New York: Wiley, c1998. ISBN 0471983373.

[3] PECHOUŠEK, Jiří. *Základy programování v prostředí LabVIEW*. 1. vyd. Olomouc: Univerzita Palackého, 2004. ISBN 80-244-0800-7.

[4] VLACH, Jaroslav, Josef HAVLÍČEK a Martin VLACH. *Začínáme s LabVIEW*. 1. vyd. Ilustrace Viktorie Vlachová. Praha: BEN - technická literatura, 2008. ISBN 978-80-7300-245-9.

[5] NATIONAL INSTRUMETS CORP. *Manuals - LabVIEW 2013 Adaptive Filter Toolkit Help.* National Instruments Corp, 2013.

[6] JONES, Douglas. L. *Adaptive Filters*. Texas, 2005. URL: < http://cnx.org/content/col10280/1.1> [cit. 2016-03-10].

[7] SCHWARZ, Daniel. *Lineární a adaptivní zpracování dat*. Vyd. 1. Brno: Akademické nakladatelství CERM, 2012. ISBN 978-80-7204-779-6.

[8] MARTINEK, R. *Využití adaptivních algoritmů LMS a RLS v oblasti adaptivního potlačování šumu a rušení*. roč. 2013, č. 1, s. 8. URL: <https://otik.uk.zcu.cz/bitstream/handle/11025/1889/r6c4c9.pdf> [cit. 2016-02-23].

[9] MALENOVSKÝ, V. *Adaptivní filtrace zašuměnych řečovych signálů*. Elektrorevue. 2002. URL: < http://www.elektrorevue.cz/clanky/02063/index.html> [cit. 2015-12-04].

[10] SAEED V. VASEGHI. *Advanced Digital Signal Processing and Noise Reduction*. 3rd ed. Chichester: John Wiley Sons, 2006. ISBN 9780470094952.

[11] MILLER, K.*Integrating LabVIEW and C#*. 2015, URL: <http://erdosmiller.com/integrating-labview-and-c-1> [cit. 2015-11-08].

[12] SMÉKAL, Z., SYSEL, P. *Signálové procesory*. 1. vyd. Praha: Sdělovací technika, 2006. ISBN 80-86645-08-8.

[13] TULACH, A. *Rozklady matic*. Bakalářská práce na Masarykově Univerzitě, Přírodovědecká fakulta. Vedoucím práce byl RNDr. Martin Tajovský. URL: <http://is.muni.cz/th/175484/prif_b/Rozklady_matic.pdf> [cit. 2015-12-04].

[14] ZHANG, F. *Matrix theory: Basic results and techniques*. 2nd ed. New York, N.Y.: Springer, c2011. Universitext. ISBN 978-1-4614-1098-0.

[15] KRAJNÍK, E. *Základy maticového počtu*. Vyd. 1. Praha: Česká technika - nakladatelství ČVUT, 2006. ISBN 80-01-03376-7.

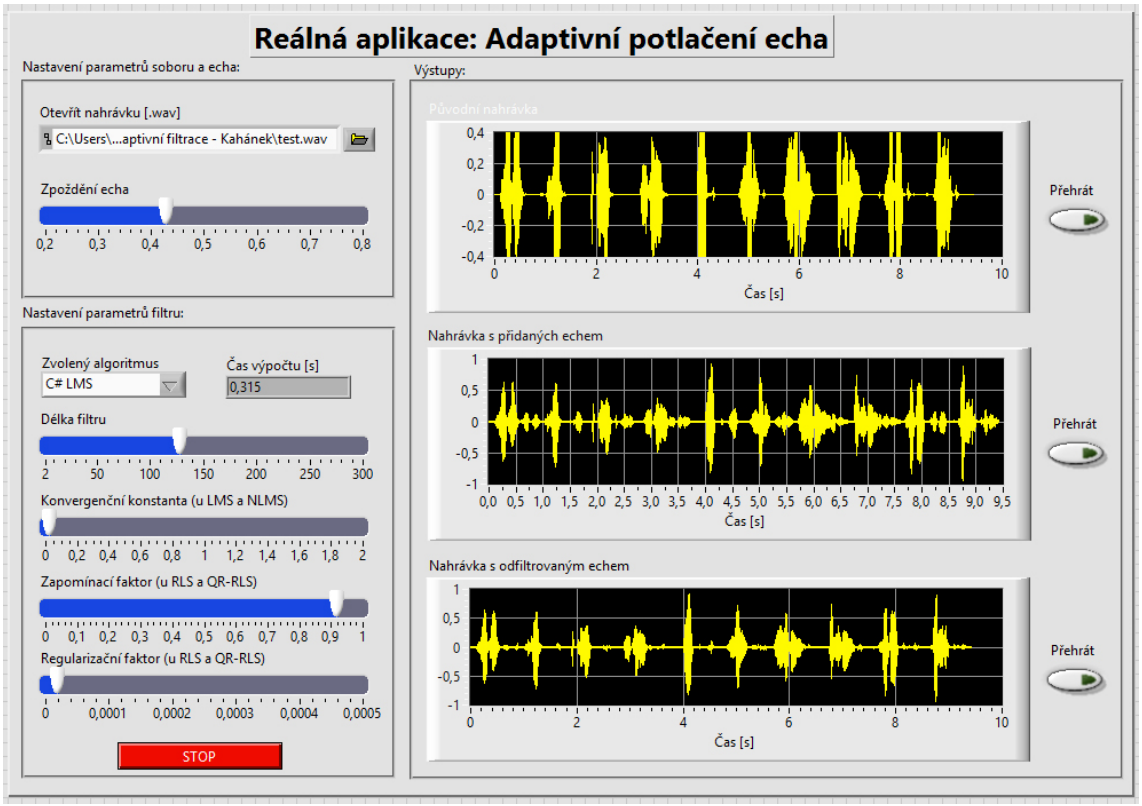
[16] DINIZ, P. S. R. *Adaptive filtering: algorithms and practical implementation*. 3rd ed. New York: Springer, 2008. ISBN 978-0-387-31274-3.

[17] BELLANGER, M. *Adaptive digital filters*. 2nd ed., rev. and expanded. New York: Marcel Dekker, c2001. Signal processing (Marcel Dekker, Inc.), 11. ISBN 0824705637.

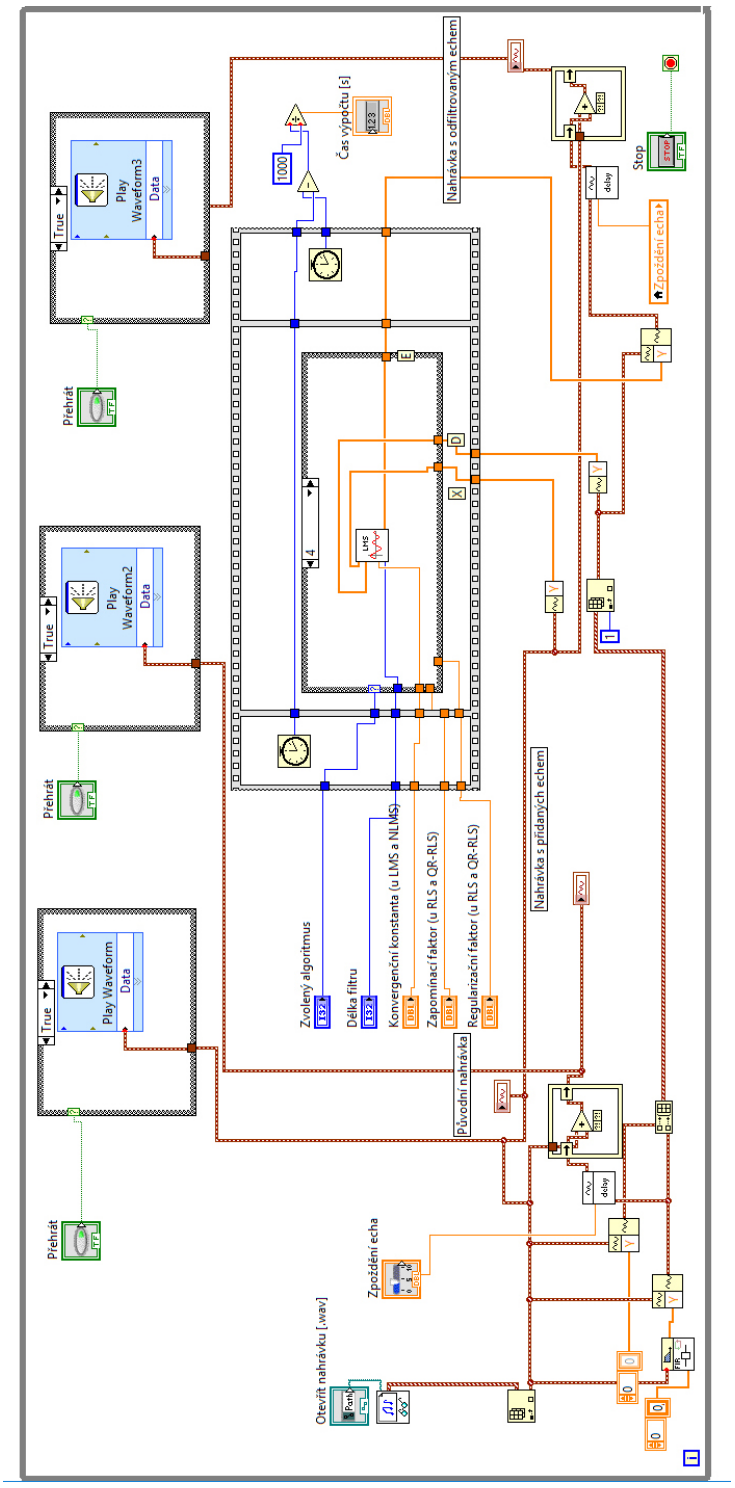
[18] POULARIKAS, Alexander D. a Zayed M. RAMADAN. *Adaptive filtering primer with MATLAB*. Boca Raton: CRC/Taylor Francis, 2006. ISBN 0849370434.

A Podoba panelů a bloková schémata aplikací

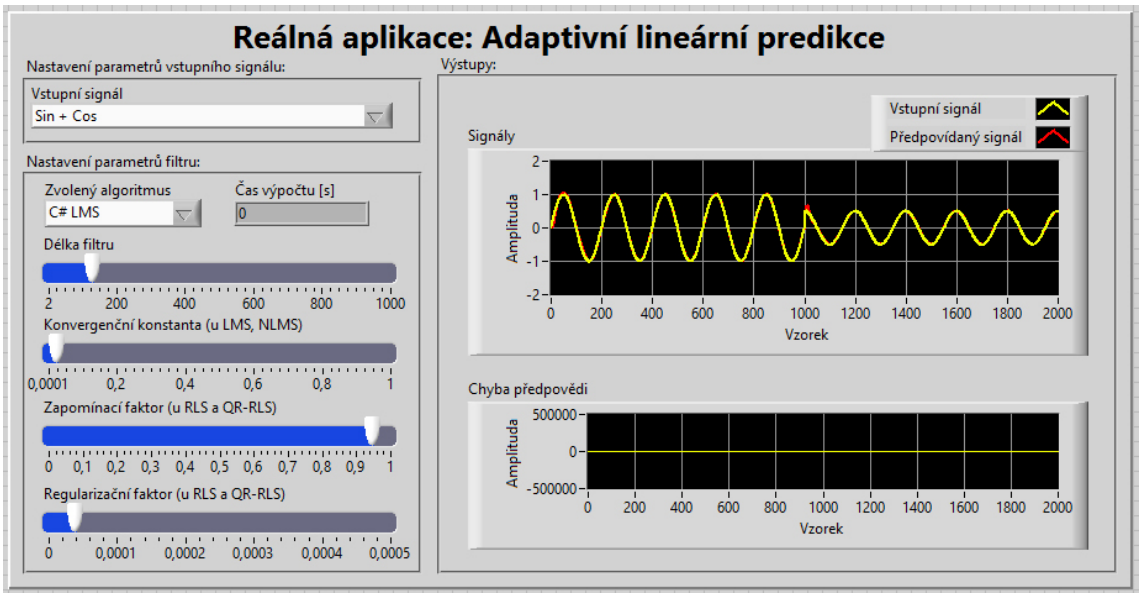
Toto je příloha ke kapitole 6. Obsahuje obrázky zobrazující čelní panely a bloková schémata jednotlivých aplikací.



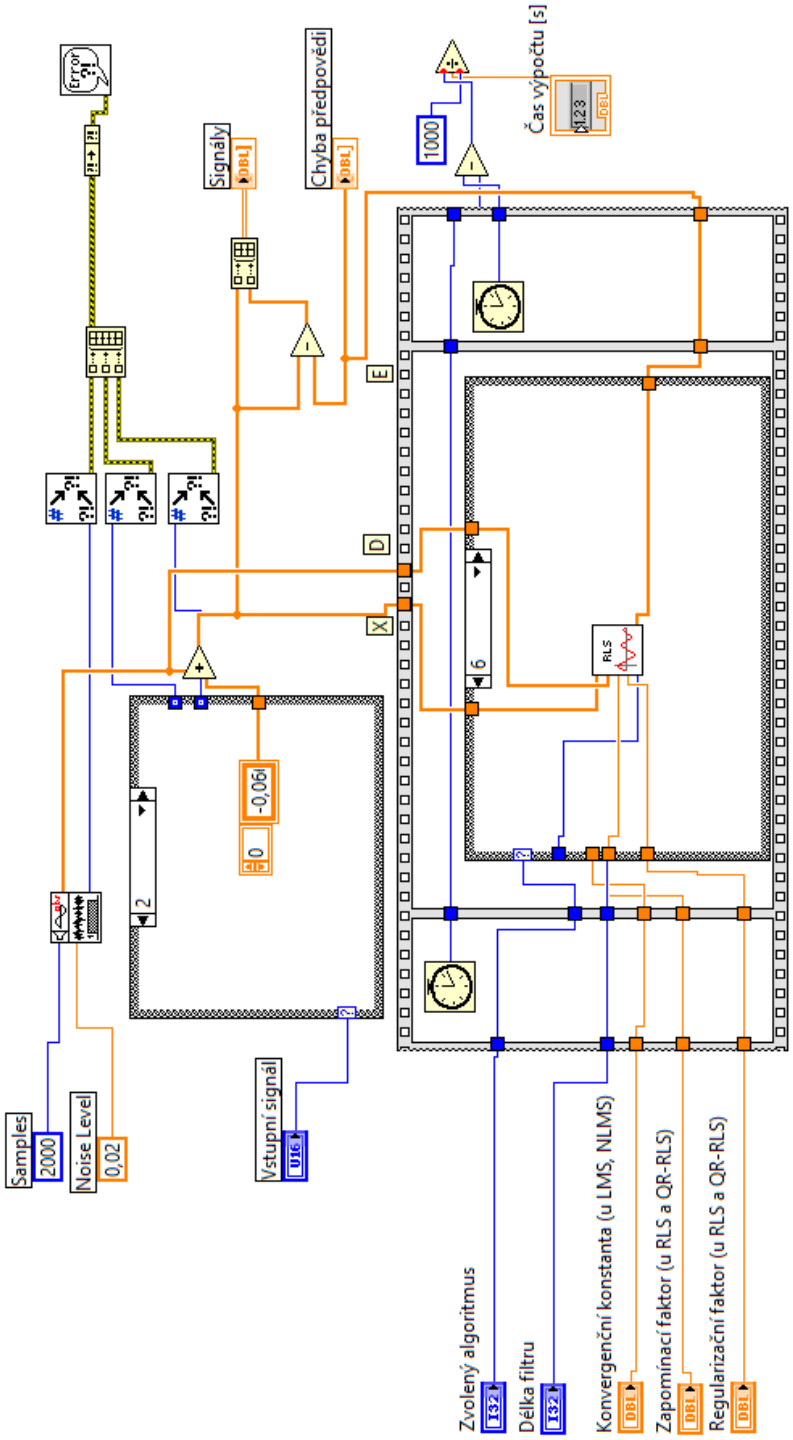
Obrázek 28: Podoba čelního panelu úlohy č. 1



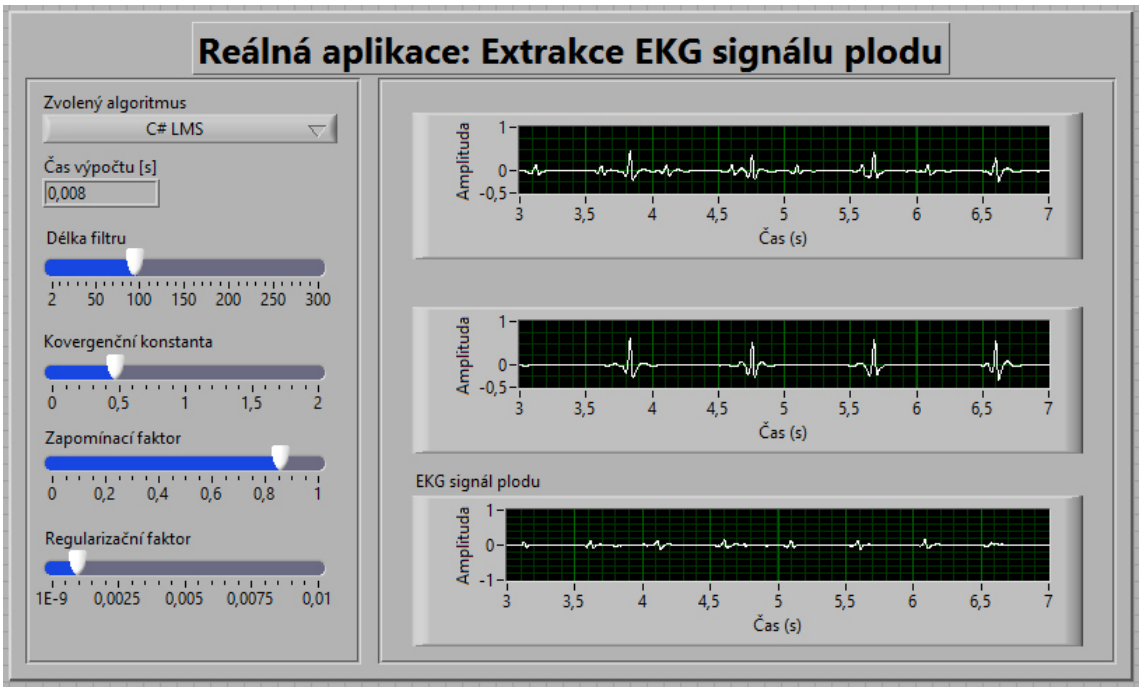
Obrázek 29: Blokové schéma úlohy č. 1



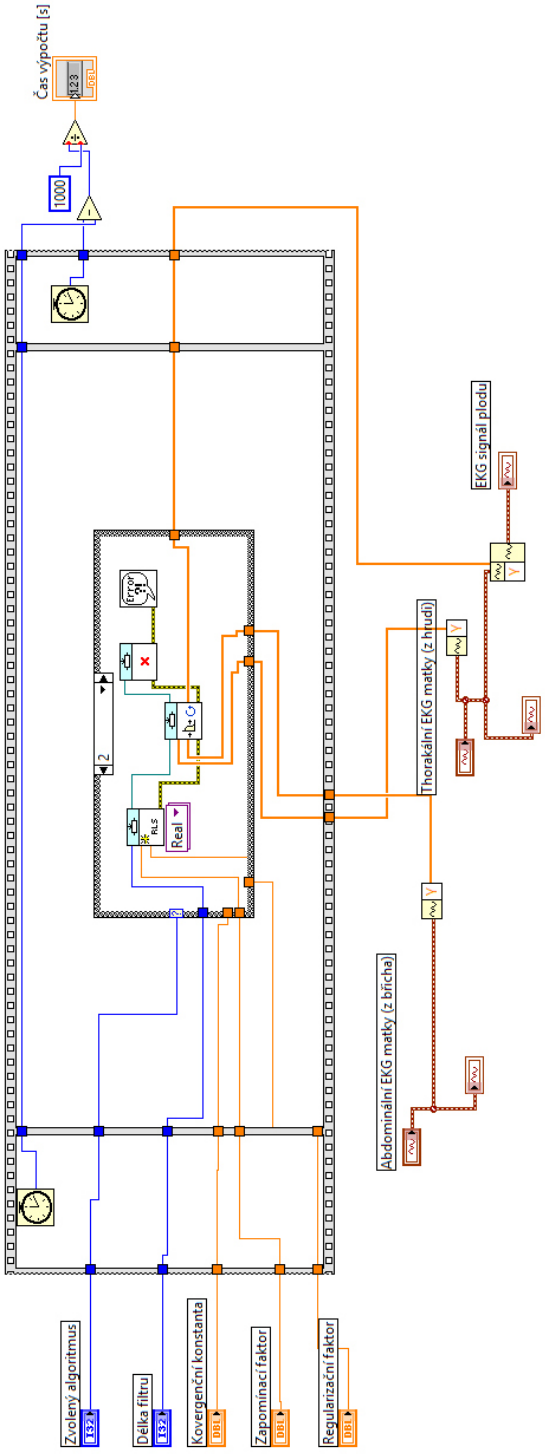
Obrázek 30: Podoba čelního panelu úlohy č. 2



Obrázek 31: Blokové schéma úlohy č. 2



Obrázek 32: Podoba čelního panelu úlohy č. 3



Obrázek 33: Blokové schéma úlohy č. 3

B Přílohy na CD

Na přiloženém CD jsou umístěny tyto přílohy:

- Realizované aplikace v programu LabVIEW (adresář KAH0019_Realizované aplikace)
- Microsoft Visual Studio projekt s implementací adaptivních algoritmů (adresář KAH 0019_ Implementace DLL knihovny algoritmů)
- Postup implementace DLL knihovny do prostředí LabVIEW (KAH0019_Implementace DLL knihovny do LabVIEW.pdf)
- Ukázka prostředí Adaptive Filter Toolkit (KAH0019_Ukázka z prostředí LabVIEW Adaptive Filter Toolkit.pdf)
- Popis paletové nabídky Adaptive Filter Toolkit (KAH0019_Popis paletové nabídky LabVIEW Adaptive Filter Toolkit.pdf)
- Dokument pro studijní učely k Úloze 1 (KAH0019_Úloha 1.pdf)
- Dokument pro studijní učely k Úloze 2 (KAH0019_Úloha 2.pdf)
- Dokument pro studijní učely k Úloze 3 (KAH0019_Úloha 3.pdf)